intel®

AP-388

APPLICATION NOTE

2

82489DX User's Manual

M. JAYAKUMAR
MULTIPROCESSING TECHOLOGY GROUP

November 1994

PRELIMINARY
Order Number: 292116-002

82489DX User's Manual

CONTENTS

CONTENTS

INTRODUCTION

82489DX is the new interrupt controller for high performance systems and 32-bit OS. Some important considerations for hardware designers are given. This application note will provide information of all registers in 82489DX and their bits and bytes organization. The control word for various programming options are given in a tabular format. Some programming hints are given to facilitate a quick understanding of the interrupt architecture and the priority model in 82489DX.

The programming model discusses the registers, their data structure like fields, bits, bytes and default register values. The system considerations and key points to be noted while programming 82489DX are discussed next. Typical examples of initialization, interrupt service routine and Spl() routines are given. The notes discuss important hardware design considerations.

Related Reference Materials

1) 82489DX Data Book, Order Number 290446.
2) An APIC based Symmetric Multiprocessor System Design AP-474, Order Number 241521.

REGISTER ORGANIZATION

The 82489DX contains both the local unit and I/O unit. I/O unit has its own Unit ID and local unit has its own Unit ID. Both units are operational at all times once they are enabled and the access can be done to both units. It should be noted that the local unit has its own version register, and I/O unit has its own version register, namely, I/O version register. The unit enable bit is provided for local unit and it is not provided for I/O unit. However, I/O unit has mask bit for each redirection table entry to mask the interrupts. Functionally I/O unit can only transmit interrupt messages whereas local unit can both transmit and receive interrupt messages. In summary, 82489DX should be viewed as an integrated chip having a local unit and an I/O unit both capable of operating at the same time.

INITIAL REGISTER VALUES AFTER HARDWARE RESET

The local unit ID register latches the value on the address pins A3 to A10 after hardware reset whereas the I/O unit ID register gets cleared to 0 after hardware reset. The local unit Version Register is cleared to 0 whereas the I/O unit Version Register contains 1111 in its Max Redir Entry field. The interrupt masks in the local timer vector table register and in the I/O redirection table entry(31:0) registers are set so that after reset all the interrupts are masked. The spurious vector register's unit enable bit is cleared so that local unit is disabled after hardware reset. Since all the interrupts are masked after hardware reset, the I/O unit will not transmit any interrupt after hardware reset until mask is cleared specifically by software and the interrupt is active.

All other registers are cleared to 0 after hardware reset.

SYSTEM CONSIDERATIONS WHILE PROGRAMMING THE 82489DX

The 82489DX register data structure contains different fields to specify the mode of operations and the options available within each mode. Since certain options are applicable to specific modes only (for example "Remote Read" mode applies only to Interrupt Command Register, it does not have any relevance to I/O unit's redirection tables) the following programming hints are provided.

82489DX and Memory Mapping

The 82489DX is a 32-bit high performance interrupt controller. It allows the CPU to do 32-bit read and write to it. By memory mapping the 82489DX, system performance can be enhanced. Even though the 82489DX can be memory mapped, its functionality as an interrupt controller should be kept in mind while programming the virtual memory management control data structure. The caching policy for the page where an 82489DX is mapped should also be done with the functionality of the 82489DX in mind. For example, the reads to an 82489DX should not be cached and writes should be write-through. Since 82489DX registers are aligned at 128-bit boundaries, memory mapping the 82489DX with interleaved memory system should not be a problem. However, it should be noted that the 82489DX does not support pipelining.

Unique ID Requirement

All the local units and I/O units hooked on an I_{CC} bus should have an unique ID before they can use the bus. This should be ensured by the programmer, since for I_{CC} bus arbitration the units (whether it is local unit or I/O unit) arbitrate with their unit ID.

PROGRAMMING THE LOCAL UNIT

Dos and Don'ts

1. The local interrupt vector table entry (and the I/O unit redirection table entry) should not be programmed for "Remote Read" Delivery mode. In other words, only Interrupt Command register supports "Remote Read" Delivery mode.
2. Local Interrupts should not be programmed with "Lowest Priority" Delivery mode.
3. Local Interrupts should not be programmed with "Reset" Delivery mode.
4. It is not recommended to use level triggered mode except for "Reset Deassert" messages.

Atomic Write Read to Task Priority Register

This section discusses issues regarding write buffer flushing and necessity of atomicity of task priority register programming.

Typically, the task priority register is written with higher priority to mask certain low level interrupts before entering into a critical section code. In a system where an 82489DX is memory mapped the CPU may buffer this task priority register write to its on chip write buffer. The following scenario can happen in such situation: CPU posts task priority register write to its on chip write buffer and enters into the critical code. A lower priority interrupt (which should not enter the critical code) interrupts the CPU before the write buffer gets flushed into task priority register). The CPU now erraneously accepts the lower priority interrupt. To avoid the situation, atomic write and read to task priority register should be done. The read following write ensures that the write buffer is flushed to task priority register and the atomicity ensures that no interrupt will be accepted by the CPU during its write to task priority. In case if the CPU itself takes care of flushing its write buffers before INTA cycle, there is no problem. However, if there are system posted write buffers then external logic should make sure to flush the system write buffers before INTA-cycle.

Task Priority Register and Total Usable Vectors

Task priority register is used to specify the priority of the task the processor is executing. In 8259 the priority is defined only among the interrupts that it handles. 82489DX goes further ahead in handling priority. In multitasking system, in addition to device interrupts, various tasks have different priority and 82489DX allows consideration of the priority at system level. The processor specifies the priority of the task it executes by writing to task priority register. Now any interrupts *at or below* the task priority will be masked until the task priority gets lowered. The masking granularity is at priority level. Out of 256 interrupt vectors 16 priority levels are specified and 16 vectors share one priority level. Since the masking granularity by the task priority register is at priority level, group of 16 vectors get masked when a local unit increases its task priority by one level.

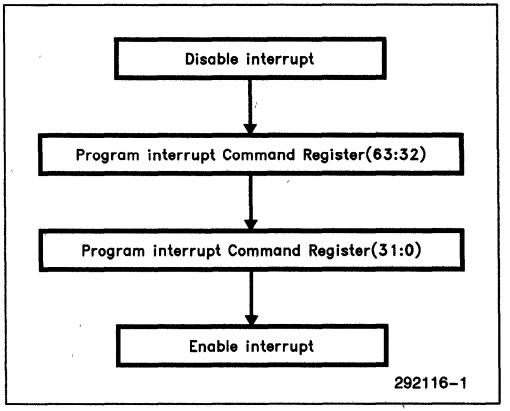
When task priority register is at its minimum level of 0, interrupt vectors having level 1 to 16 are passed to CPU. Stated in other words, even when the task priority register is at its minimum (level of 0), interrupt vectors at level 0 will be masked. This means that the interrupts should not be programmed with vectors 0 to 15. So out of 256 interrupt vectors, only 240 interrupt vectors (vector 16 to 255) can be used in 82489DX.

ISR/IRR/TMR

1. Bits 0–15 of IRR/ISR/TMR do not track interrupt. No interrupt of vector number from 0–15 can be posted. The total interrupts supported are 240. This can be easily explained by the way the priority mechanism is defined. When reading the lowest 32 bits of this register, 0 will always be returned for the lower 16 bits.

Interrupt Command Register Programming Considerations

The interrupt command register (31:0) has the side effect of sending interrupt once it is written. There is no mask bit associated with Interrupt Command Register. Once interrupt command register (31:0) is written, the interrupt is sent from the local unit. The interrupt destination is provided in the interrupt command register (63:32). So, the interrupt command register (63:32) should always be programmed before the interrupt command register (31:0) is programmed.



CRITICAL REGIONS AND MUTUAL EXCLUSION

This section discusses the reasons for mutual exclusion to be exercised when writing to interrupt command register. Each 82489DX has a single Interrupt Command Register that is used to send interrupts to other processors. The programmer should make sure to synchronize access to this register. Specifically, (1) writing all fields of the register (MSB), (2) Sending the interrupt message (by writing the LSB register), and (3) waiting for Delivery State to become Idle again, should occur as a single atomic operation. For example, if interrupt handlers are also allowed to send inter processor interrupts, then interrupt dispensing to the processor must be disabled for the duration of these activities so that interrupt handlers are excluded from accessing the ICR. This is explained as follows. Let us assume in a typical MP system preemptive scheduling (on another processor) is implemented by sending inter processor interrupts (IPIs). IPI can be also used for clock distribution in an asymmetric system where the timer interrupts only one processor and that processor notifies all other processors in the system through IPI. Inter processor interrupts are implemented by using interrupt command register. If we allow interrupts during writing interrupt command register the following erraneous operation may result. If interrupts are enabled (they should not be) during writing to interrupt command register, interrupts can come after writing to MSB portion and before writing to LSB portion of interrupt command register. Now in the interrupt service routine, if ICR is used (for distribution of interrupt to other processor(s), for example) then this ISR also starts writing to the Interrupt Command Register. That means the ISR will overwrite the MSB portion just written by the previous IPI. After returning from the ISR when the previous IPI continous writing to the remaining LSB portion, the message will be delivered to wrong address since MSB is modified by the module which interrupted. **The inference is that while accessing ICR interrupts should be disabled.** Also it should be noted that except for "Reassert Deassert Messages", IPI should only use edge triggered mode.

BUFFERING IN INTERRUPT COMMAND REGISTER

The Interrupt Command Register provides one level of buffering which should be kept in mind while programming an 82489DX. The ICR (Interrupt Command Register) becomes busy as soon as inter processor message is written into it. It hands the message over to I_{CC} bus transmit unit which in turn tries to send through I_{CC} bus. Since the ICR has passed the command to transmit Unit (whose responsibility is to send it through I_{CC} bus) it becomes free. The software before writing next inter processor message reads the flag to be free and writes next message. Thus there is a possibility of next message being written into the 82489DX before the first message is really sent out. The programmer should be aware of this.

INTERRUPT COMMAND REGISTER DO'S AND DON'TS

1. "ExtINTA" delivery mode should not be used for all destination shorthand.
2. "Remote Read" should always be programmed as "Edge" triggered interrupt.
3. "Remote Read" should always be programmed with physical Destination mode (and not with Logical Destination mode).
3. Only Fixed Delivery Mode should be used for "Self" destination shorthand. Stated otherwise, "lowest priority", "Remote Read", "Reset", "NMI" delivery modes do not apply for "Self".
4. For "All incl. self" and "All excl Self" destination shorthands, "Remote Read" delivery mode should not be used.
5. For "All incl. self" and "Self" destination shorthands "Reset" Assert mode should not be used.
6. For "All exclusive self" destination shorthand if "Reset ASSERT" delivery mode is used, it should be ensured at system level that only one processor executes this instruction at any time. To explain this, let us consider the following situation. Let us assume that two CPUs, CPU A and CPU B are executing "Reset ASSERT, All Exclusive Self". The message of CPU A puts every CPU except CPU A in reset state. After the message is written by CPU A it typically takes 2.9 μs for the message to flow through the I_{CC} bus to reach other local units to reset all other processors. Before this message resets, let us assume another processor also, say CPU B, issues the "Reset ASSERT, All Exclusive Self" message. The following CPU B message (which was sent out before CPU B itself got reset because of CPU A reset message) will reset every CPU, which will include CPU A, except CPU B. But CPU B will eventually get reset by the message sent by CPU A and CPU A will also get reset by the message sent by CPU B. Thus all the CPUs in the system goes into reset state and this is an irrecoverable state. To avoid this, only one processor should execute this instruction at any time. This can be achieved, for example, by spinlock or mutex implemented as shared variables between multiprocessors.
7. Messages could be sent out in "Logical" or "Physical" mode with destination ID of all 1's depending on the way Destination Mode entry is programmed. In brief, "All incl. self" and "All excl. self" supports both "Logical" and "Physical" addressing mode.
8. When destination shorthand (*i.e.*, broadcast) is used with "lowest priority" destination mode, then even though all participates in arbitrating for destination, only the lowest priority gets the message. So even though the addressing is broadcast since the destination mode is lowest priority only one gets the message.

9. When destination shorthand (*i.e.*, broadcast) is used with "Fixed" destination mode, then all the units get the message. So to send messages to all units Fixed destination mode should be used in addition to using destination shorthand.
10. It is recommended that all IPI messages, except for "Reset Deassert", use only edge triggered interrupt mode.

IPI THROUGH INTERRUPT COMMAND REGISTER

Interrupt command register can be used to send inter processor interrupt. Inter processor interrupts can be used for preemptive scheduling, TLB flushing, clock distribution, etc. IPIs can also used in asymmetric systems to pass certain work to another processor who has exclusive access to certain piece of hardware. Let us consider a dual processor system which uses only two 82489DX in the whole system. Since the local units should be accessible only by their respective processor a local unit should be selected only when the arbitrator grants the bus to its processor. Since 82489DX has common chip select for its local unit and I/O unit, for logical simplicity, system hardware may select a 82489DX when the corresponding processor is granted bus. Because of this, processor A can access only I/O unit and local unit that are available in its 82489DX. It is not possible to access the I/O unit of the other 82489DX. The same thing holds good for the other processor. Since I/O unit should be globally visible to both processors, there may be situations when a processor may want to access the other I/O unit. This is typically the case for enabling and disabling the I/O interrupt. IPI can be used to pass that task to the other processor which can access that I/O unit. This is just one example for using IPI.

ExtINTA INTERRUPT POSTING

ExtINTA interrupts are used to support 8259 in a 82489DX based system. The external interrupts (ExtINTA) are specific in their characteristics in that they do not have any priority relationship with rest of the interrupt structure. But when posting an interrupt to the processor, if both an external interrupt and a 82489DX interrupt are pending, 82489DX could post either one to the processor. In 82489DX implementation, it would post external interrupt whenever there is no other 82489DX interrupt that can be posted to the processor. It should be also noted that External Interrupts can not be masked by raising task priority. However, they can be masked by the mask bit in the table entry for that (ExtINTA) interrupt.

Since ExtINTA interrupts do not have any priority relationship, ISR and IRR bits are not maintained for external interrupts. As far as interrupt acceptance is concerned, if more than one ExtINTA interrupts are directed towards a local unit, that local unit treats all the ExtINTA interrupts directed to it as only one ExtINTA interrupt. This leads to an important point that in a system no more than one interrupt should be programmed as ExtINTA interrupt type with the **same destination**. However, it should be noted that there can be more than one ExtINTA' type of interrupt in a system with each having different local unit as destination.

LOWEST PRIORITY

Under the lowest priority delivery method, the processor to handle the interrupt is the *one* in the specified destination with the lowest processor priority value. If more than one processor is at the lowest priority, then a unique arbitration ID is used to break ties. To have unique arbitration ID in the system (which is mandatory for the lowest priority algorithm to work) all the arbitration ID of local 82489DXs in the system should be in sync. On reset, arbitration ID is reset to zero by the hardware. Hence all the local units in the system after reset will have same arbitration ID (namely zero). For lowest priority arbitration to work properly we need to have unique arbitration ID in the system. This means after local unit IDs are written in all local units (obviously, each unit ID should be different from other IDs) a RESET DEASSERT message should be sent in ALL INCLUSIVE mode. The important side effect of RESET DEASSERT message is that it copies the unitID into the respective arbitration ID. Since unit IDs are unique, the RESET DEASSERT message ensures that the arbitration ID also are unique in the system. This RESET DEASSERT message should be sent before system is used for lowest priority arbitration.

The RESET DEASSERT message, if not sent, only once delivery semantics may not be guaranteed. If RESET DEASSERT message is not sent then all the arbID in the system will be same. When a message is sent in the lowest priority arbitration, the participating local units use their processor priority concatenated with arbitration ID to decide the destination. Processor priority is derived from the task priority. There is a chance that two local units can have same task priority depending on the code they are executing and thereby same processor priority. In addition since arbID are also same if RESET DEASSERT message WAS NOT sent, all the processors in the same priority may accept the message in lowest priority arbitration. This violates the only once delivery semantics. The inference is that RESET DEASSERT message in ALL INCLUSIVE SELF mode should be sent as part of initialization before enabling interrupt in the lowest priority destination scheme.

It should be noted that only once delivery semantics for a group destination is guaranteed only if multiple fixed delivery of the same interrupt vector are not mixed.

DISABLING LOCAL UNIT

Once the 82489DX is enabled by setting bit 8 of spurious vector register to 1, the user should not disable the local unit by resetting the bit to 0. The result will put the local unit in an inconsistent state. However, a local unit can be disabled by getting "reset" interrupt message from any other local unit across the I_{CC} bus.

ISSUING EOI

EOI, End of Interrupt issuing indicates end of service routine to 82489DX. Always the highest priority ISR bit which is set during INTA cycle gets cleared by EOI. This section discusses the relevance of EOI to the specific types of interrupts and its timing related to interrupt deassertion.

EXTERNAL INTERRUPTS AND EOI

External Interrupts (ExtINTA) should be programmed as edge type. INTA cycles to external interrupts are taken automatically as EOI by 82489DX. This is similar to AEOI, Automatic End of Interrupt of 8259A. So EOI should not be issued to 82489DX for ExtINTA interrupt servicing. For ExtINTA type of interrupts, there is no need to have interrupt service routines since the main purpose of ExtINTA interrupt itself is to have software transparency in the compatible mode. The existing interrupt service routines written for 8259 will be executed by the processor for ExtINTA interrupts.

SPURIOUS INTERRUPTS AND EOI

Spurious Interrupts do not have any priority relationship to other interrupts in the system. So IRR is not set for spurious interrupts. EOI should not be issued for spurious interrupts. It is advisable not to share the spurious interrupt vector with any interrupt.

If spurious interrupt vector is shared with some other interrupt then the following guidelines should be followed. If the source is spurious interrupt (for which the corresponding ISR is **not** set) then EOI should **not** be issued. If the source is a valid interrupt sharing the spurious interrupt vector (for which the corresponding ISR is set) then EOI should be issued.

NMI AND EOI

For NMI type of interrupt no IRR bit is set. So, obviously EOI should not be issued while servicing NMI type of interrupts.

PROGRAMMING I/O UNIT

Interrupt Sharing Considerations

Two different interrupts should not be programmed with the same interrupt vector. This means that each redirection table in a system should have unique vector. Interrupt sharing can be done electrically. Interrupts connected at different interrupt input pins of 82489DX CAN NOT share interrupt by having same vector. 82489DX does not support active low interrupts. So sharing interrupts should have polarity logic support externally.

I/O Unit and Priority

The 82489DX partitions its interrupt control function among two different units:

1. I/O unit
2. local unit

The priority resolving is done at local unit. The I/O unit does not involve itself in the priority mechanism. The I/O unit takes a snapshot of interrupts pending at the INTIN interrupt input pins. If interrupts are active, it starts sending the interrupt messages over I_{CC} bus. It starts sending the lowest numbered interrupt input first. That is if INTIN0 and INTIN5 are found active in a snapshot, interrupt message corresponding to INTIN0 is sent first regardless of the priority of the vectors that are associated with these interrupts. It sequentially sends all the interrupts found active in a snapshot. Before sending, it checks whether the corresponding INTIN is still active. **This is the reason why interrupts, both edge and level triggered, should be kept active until CPU acknowledges it.** The difference between edge triggered and level triggered interrupt is that edge triggered interrupts ensure only one activation of interrupt per low to high edge whereas the level triggered interrupt allows to have multiple interrupts as long as the interrupt is held high. It should be noted that both edge and level triggered interrupts are active high.

MP SYSTEM

Initialization Sequence

This section assumes the system with multiple CPUs with each CPU having its own 82489DX local units and local interrupts (like local secondary cache data parity interrupt, coprocessor interrupt etc.,) connected to the respective local units. The system additionally assumes symmetric multiprocessing in the sense that I/O system is symmetric and it can be initialized by any CPU in the system.

Section A: Code Executed by all CPUs in the system

Section B: Synchronization to indicate Section A is completed

Section C: Only one CPU need to execute this Code

Each local unit is visible (through address mapping) only to that CPU to which local unit is attached. So each local unit will be programmed by its own CPU. Thus the code specified as section A will be executed by all CPUs in the system.

Section B of the initialization code is also executed by all the CPUs. This section of the code ensures that all the CPUs have completed execution of their "Section A" so that all Local units are properly initialized with different IDs, the system is in a consistent state, etc.,

Section C initializes system wide I/O unit and enables the interrupt mechanism to start functioning. Since the I/O unit is system wide, only one CPU need to program the I/O unit part of the 82489DX.

Section A

Write the local Unit ID (if needed)

Write all Ones to Destination Format Register

Write Logical Destination Register

Raise the Task Priority

Program the Spurious Interrupt Vector
Vector and Enable the Local Unit

Program the Vectors for Local Interrupts and Timer

Program the Timer Control Registers

Clear the mask for Local Interrupts and Timer

Initialize the local Interrupt sources

Broadcast ALL INCL. SELF
Reset DEASSERT message

Lower the Task Priority

It should be noted that the interrupt descriptors, interrupt service routine, spurious interrupt service routine and other interrupt related structure should have been initialized before the Section A. This is because Section A code enables respective local interrupts and timer interrupt vectors and when the interrupts arrive from these devices Section A ensures that 82489DX will provide the vector. But the code executed before Section A should ensure the interrupt structure is initialized. Spurious interrupts are bound to occur because of the asynchronous interaction between interrupts and software writing to task priority register. So spurious interrupt service routine has to be initialized before Section A.

WRITE THE LOCAL UNIT ID (IF NEEDED)

Each local unit can get the ID latched by reset from 82489DX address pins A3–A10. If the hardware ensures that during reset each local unit in the system gets different pattern on the address pins A3–A10 then all the local units are initialized automatically with different IDs. In that case writing to the local unit ID by software is not mandatory. If software writes the local unit ID then it should be read from some address space which is same for all CPUs but have different IDs for different CPUs. This will ensure that the same code when executed by different CPUs will initialize respective local unit with different ID.

WRITE ALL ONES TO DESTINATION FORMAT REGISTER

All the 32 bits of Destination Format Register are written with 1. This is to support single level logical addressing mode. This mode is explained in the following paragraph.

WRITE TO LOGICAL DESTINATION REGISTER

The logical Destination Register should be written with the logical destination address. It should be noted that since each CPU needs to assign a different logical Destination address to its own 82489DX local unit and since this code is executed by all CPUs the logical Destination address should be read from some address space which is the same for all CPUs but contains different Destination address values. Since logical Destination address is in bit decoding format, typically this can be achieved by shifting the CPUID.

The logical destination register with Destination format register can be used to support flat model. In this model, bits 24 through 31 of the destination address of the interrupt message vector are interpreted as decoded field. Intel strongly recommends for future compatibility to use only bits 31 to 24 of logical destination register. To have binary compatibility with future APIC implementations, any code written for 82489DX should not use bits 0 to 23 of logical destination register. This field is compared against the logical destination register of the local unit. If there is a bit match (i.e., if at least one of the corresponding pair of bits of the destination field and logical destination register match) this local unit is selected for interrupt delivery. Each bit position in the destination field corresponds to an individual local unit. For future compatibility, only bits 0 to 23 of logical destination register should be zero. This scheme allows the specification of arbitrary groups of 82489DXs simply by setting the member's bit to one, but allows a maximum of 8 local units in the system since bits 0 to 23 of the logical destination register is zero. Broadcast to all is achieved by setting all 8 bits of destination to ones. This selects all 82489DXs in the system.

If more than 8 units are to be addressed in the system (and if future compatibility is not a major issue) then all the bits of the logical distination register can be used as a bit map thereby increasing the number of CPUs addressable in logical addressing to 32.

RAISE THE TASK PRIORITY

Before enabling the local interrupts and timer interrupts the task priority is raised to maximum priority in the system so that these interrupts are masked temporarily.

PROGRAM THE SPURIOUS INTERRUPT VECTOR AND ENABLE THE LOCAL UNIT

The spurious interrupt vector register is programmed with the corresponding vector. This vector will be pointing to a dummy routine with just an IRET. The unit is enabled so that the tristate pin PINT can come out of tristate state to pass the interrupts.

PROGRAM THE VECTORS FOR LOCAL INTERRUPTS AND TIMER

The interrupt vectors for Local Interrupts and timer are initialized with corresponding vector.

PROGRAM THE TIMER CONTROL REGISTERS

The timer registers such as divider configuration register, initial count, mode of operation and source of the timer clock are programmed.

CLEAR THE INTERRUPT MASK FOR TIMER AND LOCAL INTERRUPT

The interrupt mask is cleared for timer and local interrupts by clearing the interrupt mask bit in their respective interrupt vector register.

INITIALIZE THE LOCAL INTERRUPT SOURCE

The local interrupt sources are also programmed for proper system operation. This involves enabling the interrupt from the sources. The order of enabling the interrupt is very important. First the 82489DX entries should be cleared and then the sources connected to the pins should be enabled. If done the other way, interrupts may get lost. This is true particularly in edge triggered interrupt inputs where if 82489DX mask is cleared after enabling the source interrupt, 82489DX may not have a chance to capture the low to high edge which might have produced immediately after the source interrupt is enabled and before 82489DX mask is cleared.

BROADCAST ALL INCL. SELF RESET DEASSERT MESSAGE

This is done so that all the local unit's ArbIDs are in sync. It should be noted that for breaking the tie during lowest priority arbitration ArbID is used. ArbID is copied from local unit ID during reset. Since local unit IDs can be written through software and at that time ArbID is not updated there may be a case where all ArbID in a system to have same value. To avoid such situation Reset Deassert message is sent to ALL INCL. SELF so that the ArbIDs are different in the system.

LOWER THE TASK PRIORITY

Task priority is lowered so that the interrupts can be armed to the CPU.

Section B

Synchronization

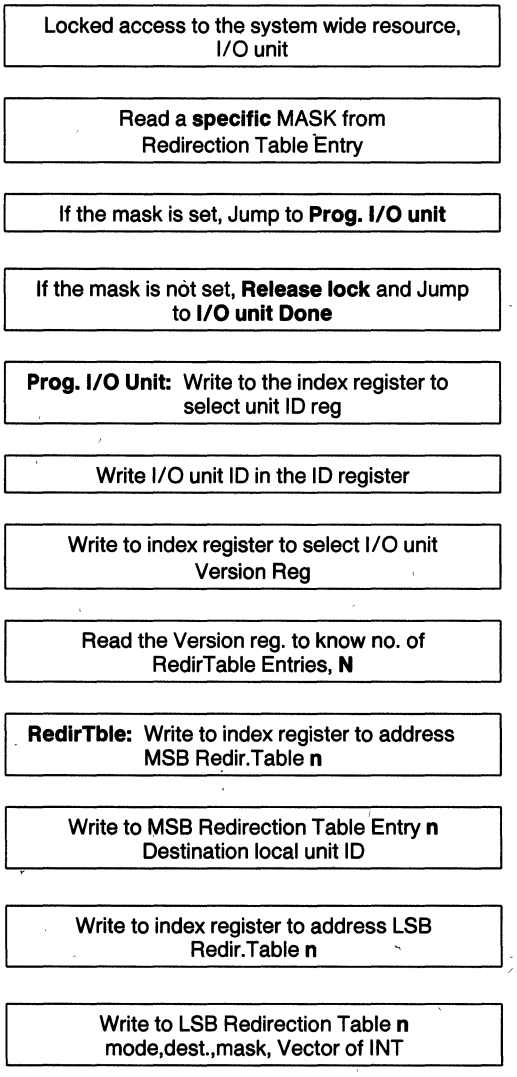
There are many methods available for synchronization. **Test - and - set** is a simple primitive, for example, available for synchronization. **Counting semaphores** can be built using this test - and - set primitive and synchronization can be achieved.

The main idea is to achieve global synchronization among the processors to indicate the local unit portion is programmed.

Section C

SYSTEM WIDE RESOURCES PROGRAMMING

This portion needs to be programmed by one CPU only. It should be noted that since the system environment we are assuming is shared memory symmetric MP system, CPU specific coding is not possible. System wide resource programming can be achieved by many ways depending on simplicity and performance (since this is only initialization routines, performance should not matter much) tradeoff. The following sequence illustrates a simple approach to program. The assumption here is that 82489DX will get reset both during cold reset and warm reset.

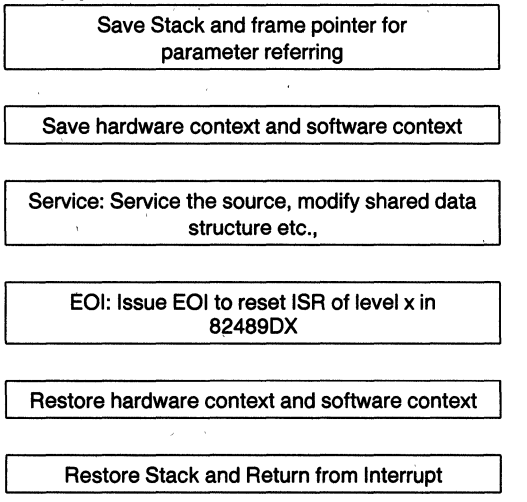


The first CPU getting the lock will find the mask to be set (since after reset, 82489DX mask is set). It locks the I/O unit for programming. The mask selected is specific in that the system initializes such that the mask is cleared during initialization. So by reading that mask mutual exclusion is achieved. If the system requirement is such that no mask can be cleared during system initialization some other register can be read. For example, the I/O unit IDs are reset to 0 on reset. Since the system initialization will have all the local unit IDs starting from 0 and I/O unit will be initialized by the system to non Zero ID, I/O unit can be read and if 0 can be assumed that programming is not yet done (so that it can gain control of lock and start programming) and if found non Zero, then that CPU can skip programming the I/O units by jumping to I/O unit done.

The I/O unit registers are organized as index register and data register. Other portions of section C are self explanatory. After I/O unit done, the I/O system initialization can be started so that the interrupts can start flowing in the system.

INTERRUPT SERVICE ROUTINE

ISR (x)



The above ISR x shows the interrupt service routine of interrupt level x. The stack, hardware context like CPU registers and software context like task specific variables are saved. The Servicing is done as specific to the interrupting source. This may involve reading a status register or initiating a thread to read a "full" buffer, initiating a thread to write data to some "empty" register, or acknowledging an interrupt from another CPU. This is the point at which the interrupting source is supposed to deactivate its request. Next EOI is issued to reset the ISR bit corresponding to the interrupt level x. Till now the interrupts from same level and lower levels were masked. Once EOI is written, interrupts from all the levels can start coming. The hardware context and software context are restored followed by stack cleaning and a proper Return from Interrupt is executed.

There are couple of timing issues that can be considered here. The time delay between **Service** and **EOI** is referred here. This timing and its relevance to edge/level triggered interrupt is discussed as follows: In the case of edge triggered interrupts, for each edge one Interrupt message is sent by I/O unit to local unit over I_{CC} bus whereas for level triggered interrupts there are two interrupt messages sent, one during assertion of level interrupt, and another during deassertion of level interrupt. In edge triggered interrupts, since the deassertion of interrupt does not result in any interrupt message, there are not many issues with the timing delay between **Service** and **EOI**, even though in general delaying EOI means interrupts from the same interrupt source are kept pending from interrupting CPU. In level triggered interrupts after **service** the I/O device starts deasserting its interrupt request. This results in an interrupt message to clear IRR bit in the local unit. This may take some time because the minimum possible time in I_{CC} bus is 2.3 μs (10 MHz I_{CC} clock assumed). If the I_{CC} bus is occupied by some other messages already then this IRR clearing message has to wait to get its turn which means additional delay. If EOI is issued before this happens then ISR gets cleared and IRR for this "done interrupt" is still alive to erroneously set ISR again. This will result in another interrupt. So "Early Servicing" is advisable in level triggered interrupts.

DOS Environment

In the DOS environment the initialization portion is the only routine to be coded since the 82489DX acts as a virtual wire once initialized and needs no more programming. Since it is uniprocessor environment there is no need for synchronization.

The interrupt from 8259 is programmed as type ExtINTA and other redirection table entries are not accessed since their masks are set by reset and hence disabled.

In the Interrupt service routine, since EOI is not needed for ExtINTA type of interrupts, no programming is needed for 82489DX. Since ExtINTA type of interrupts do not have any relationship to task priority, Spl routines do not apply for DOS configurations.

Transition from 8259 to 82489DX

Typically, platforms with 82489DX will support 82489DX in virtual wire mode. The BIOS in the EPROM will program the 82489DX in "Virtual Wire" mode. Typically systems boot DOS and then the 32 bit high performance OS is given control. There are also situations where after BIOS code is executed the high performance OS is given control. In both the situations, the 8259 will be operational during the DOS or BIOS portion of the code and interrupts will be flowing in the system. When the high performance OS is given control, it may want to disable the interrupt during initialization. This will involve disabling 8259. After disabling 8259, the 32 bit OS initializes and then it may want to enable interrupt mechanism which involves enabling 82489DX. The sequence we are encountering here is 8259 (and one input of 82489DX enabled in "ExtINTA" mode) enabled, 8259 disabled and then 82489DX enabled. When 82489DX is enabled in 32 bit OS all the interrupt inputs are enabled as opposed to the only one interrupt enabled in "Virtual Wire" mode. The additional difference is that 82489DX is no more a "virtual wire" but it is functioning as an interrupt controller.

In the above situation, consider the following scenario. The 82489DX is functioning as "virtual wire" and passing the 8259 interrupts as "ExtINTA" mode to the local unit. When interrupt mechanism is disabled by CLI (Clear interrupt) or masking the 8259 interrupt, there may be a possibility that already 8259 originated interrupt may be pending at the local unit asserting interrupt to the CPU. Now since the CPU has executed CLI, the interrupt is not serviced and the interrupt is kept pending. It should be noted that the pending interrupt is of type "ExtINTA". After this, 32 bit OS gets loaded which configures 82489DX redirection tables and interrupt is enabled. Now the "old pending" interrupt is delivered and since it is "ExtINTA" the external hardware will typically pass the interrupt acknowledge cycle to 8259. But at this point of time 8259 has been masked by 32 bit OS. Hence the "masked" 8259 responds with IR7 vector. So the 32 bit OS should reserve IR7 vector for both master and slave 8259 for "emptying" the old pending interrupt since the "Virtual Wire" remembers the previous interrupt.

Sequence of Enabling: In the case of enabling interrupt controllers in "ExtINTA" mode the 82489DX should be enabled before the 8259 interrupt Controller is enabled. This is because ExtINTA is "edge triggered" and if 8259 is enabled before 82489DX, 8259 might have

given an interrupt request by activating its interrupt output while 82489DX is still not enabled. When 82489DX is enabled the interrupt input has a high level and 82489DX had no chance of capturing the low to high edge of 8259.

Spurious Interrupt Service Routine

It is advisable not to share spurious interrupt vector with any genuine interrupt source. This section assumes that spurious interrupt vector is not shared with any other interrupt.

82489DX does not set ISR in response to spurious interrupt, NMI type of interrupt, Reset type of interrupt and ExtINTA type of interrupts. For all these interrupts EOI should not be issued.

Some systems have a variable count in the supervisor data structure to count number of spurious interrupts raised in the system. This can be used to study the reliability and "noise level" of the system. But in 82489DX architecture, spurious interrupt can occur even by a dynamic write to task priority register, frequency of spurious interrupt does not mean anything related to "noise level".

Return from interrupt

Spl(x) Routines

The processor handles the I/O system through device driver interface. The device driver consists of two entries to access the I/O system: 1) Call entry and 2) Interrupt entry. The interrupt entry is the one that we have been discussing for a while, i.e., interrupt service routine. The Call entry is the way the I/O system is accessed to initiate and service devices. The call entry has its own task priority and interrupt entry has the priority that is associated with the device interrupt level. The call entry and interrupt entry processes have I/O data structure like linked list, buffer pointers in common which they share. Mutual exclusion is needed to ensure the integrity of I/O system.

To ensure the mutual exclusion between these two processes running in the same processor, Spl(x) routine is used. The call entry routine (which is normally at a lower priority than the interrupt entry routine) calls Spl(x) routine to elevate its own priority above (or equal to) that of the corresponding device's interrupt priority. At this priority the interrupts from the device are masked out and the shared data structures can be accessed (exclusively).

Once this is done the priority is restored back to original value so that other interrupts won't suffer for relatively long time.

Spl() is used to save the current task priority and Spl(x) is used to elevate the task priority.

Spl()

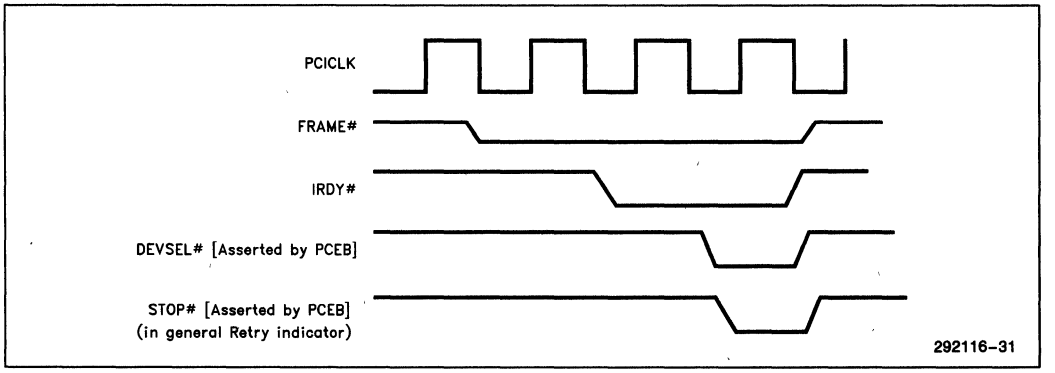
Read and return the 82489DX local unit task priority register

Spl(x)

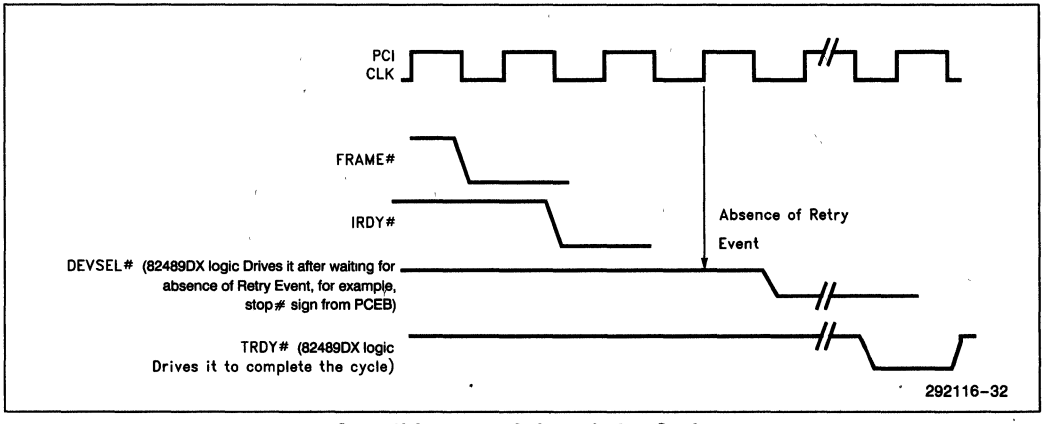
Write x to task priority register to raise priority to x

Typical usage of these routines

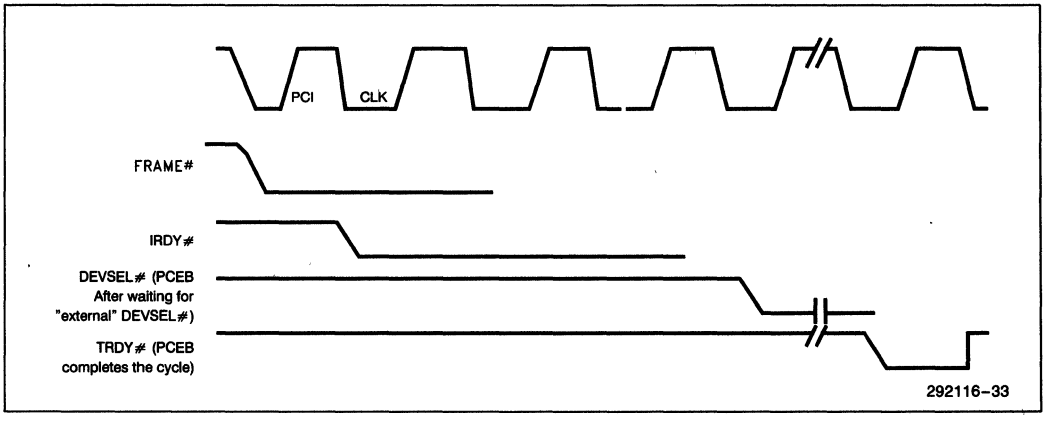
```
y = spl()    //Save the current task priority register value//
spl(x)       //Raise the task priority value                 //
:
:
;            // Access the shared data structure             //
spl(y)       // Restore the task priority register           //
```



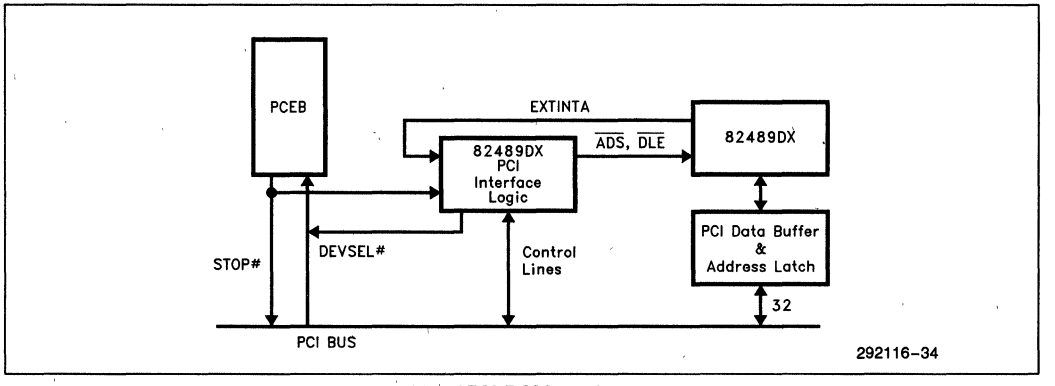
Case 1) Any INTA# Cycle Buffer Management and Retry



Case 2) Interrupt Acknowledge Cycle
Source of the Interrupt and Vector is 82489DX



Case 3) Interrupt Acknowledge Cycle
Source of the Interrupt and Vector is ESC 8259



82489DX-PCI Interface

82489DX AND PCI-EISA BRIDGE INTEROPERABILITY

82489DX gives performance benefits in multithreaded operating systems. Thus in both uniprocessor and multiprocessor systems 82489DX enhances the system level performance. This is because of its advantage in task priority management. Intel's PCIset EISA bridge component PCEB (B-stepping onwards) can interoperate with 82489DX. In this section, we will go over the "hook" provided by PCEB to connect 82489DX in a PCI system with some external glue logic.

From the Interrupt acknowledge timings of PCEB (B-stepping onwards) it can be inferred that **whenever the internal data buffers are empty**, on an interrupt acknowledge cycle, PCEB waits one clock cycle so that PCI interface logic for 82489DX can activate DEVSEL#. But, if the internal data buffers are not empty, then PCEB drives STOP# to retry the INTA cycle so that it can flush the buffers.

If DEVSEL# is seen active and if the PCEB's internal data buffers are empty, then PCEB allows 82489DX to own the INTA cycle. Thus, the external 82489DX glue logic, on an INTA cycle, should first sample STOP#. If STOP# is driven, then the glue logic should ignore the cycle. Because PCEB has some data in the buffers it wants to flush them before INTA cycle is run. So, the 82489DX glue logic should not start the cycle to 82489DX. If STOP# is not active and if the "ExtINTA" pin from 82489DX is inactive (to indicate that the cycle is for 82489DX) then it should drive DEVSEL# immediately to own the INTA cycle. At the same time it can start the cycle to 82489DX. It should be noted that 82489DX needs two INTA cycles whereas PCI bus has only one INTA cycle. So, the external logic is responsible for splitting one PCI INTA cycle into two 82489DX INTA cycles back to back but pass only one READY to the system.

During INTA cycle on PCI bus if "ExtINTA" pin is active (to indicate that it is 8259 INTA cycle), then the 82489DX glue logic should not drive DEVSEL#. Thus by finding DEVSEL# inactive, the PCEB will respond to the INTA cycle.

Thus with minimal external glue logic, it is possible to design an APIC based PCI system. Since PCI local bus will improve the I/O performance of the system, APIC will enhance the improvement of the overall system performance in a multithreaded environment.

HARDWARE DESIGN CONSIDERATIONS

Design Consideration 0

Any edge triggered interrupt creating an active edge while the interrupt is masked at 82489DX is lost. The 82489DX samples the edge triggered interrupt input only when it is unmasked. If an edge occurs while the interrupt is masked, that interrupt is lost. The software should always unmask the interrupt at 82489DX and then enable at the device. By this, it is made sure that 82489DX will have a chance to find the active going edge.

Design Consideration 1

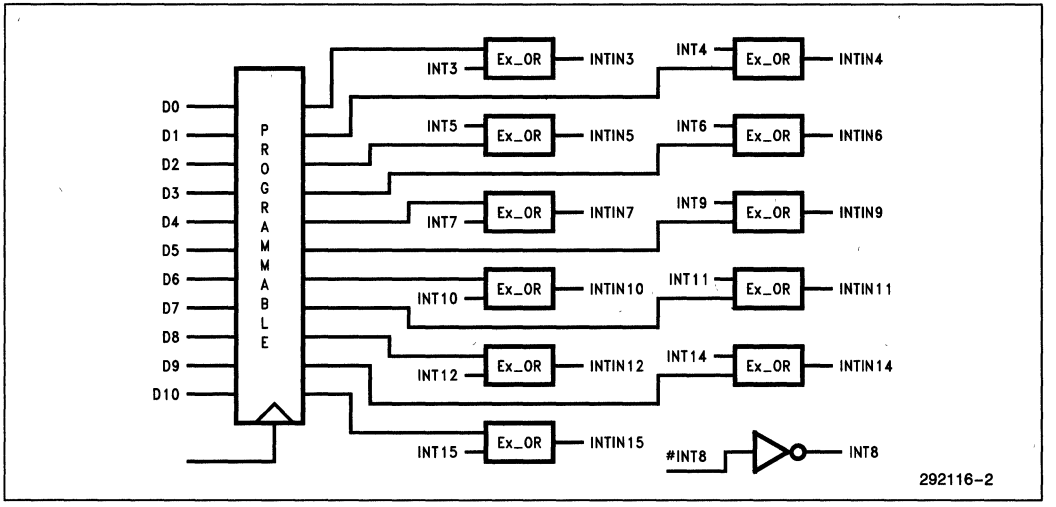
Description: The following design consideration has to be taken care of when using ISP (82357) as external interrupt controller. 82489DX allows connecting external 8259 type interrupt controller at one of its inputs. The mode associated with the interrupt input which has 8259 connected to it is called ExtINTA mode. 82489DX allows only EDGE TRIGGERED program-

ming option for ExtINTA mode. But in the case of 82357, the INT output from ISP stays high in case more than one interrupt is pending at its inputs. It does not always inactivate its INT output after INTA cycle. This will lead to a situation where ISP keeps the interrupt at high level continously and waits for INTA cycle. But since 82489DX expects an edge for interrupt sensing (for ExtINTA interrupts) it does not pass the interrupt to CPU and further interrupts are lost. So External circuitry should monitor the end of SECOND CYCLE of INTA cycle and force an inactive state at 82489DX's input. This can be done by ANDing ISP's output with a forced brief low going pulse at the end of second INTA cycle. This will generate an edge for each interrupt at 82489DX's input. For more refined edge generating logic, refer to data book, Order Number 290446.

Design Consideration 2

Description: The following design consideration has to be taken care of when using 82489DX in EISA systems. EISA ISP(82357) chip integrates 8259A. It additionally allows sharing of interrupts. To facilitate this sharing it has a programmable register, ELCR (Edge / Level trigger control register) by which certain interrupt inputs can be programmed as edge (low to high except for RTC) or level (the level is active low). The determination of edge or level is done during initial configuration of EISA system by reading EISA add in boards from the interrupt description data structures. The solution is to have programmable logic at the interrupt inputs so that 82489DX is compatible with EISA ISP. This will introduce one more register and logic to support this. This should be an 11 bit programmable register and an array of ExOR logic (12 ExOR gates or equivalent PLD). The ISP allows programmability of the following interrupts. It is highly recommended to use the same address of ELCR (and also bit definitions) for this polarity register, if possible, so that when ELCR is written this register will also be written. By this there is no separate programming needed for this polarity register. This will help to maintain compatibility with future APIC implementations which may use the existing ELCR register itself for polarity control. This is true for integrated APIC.

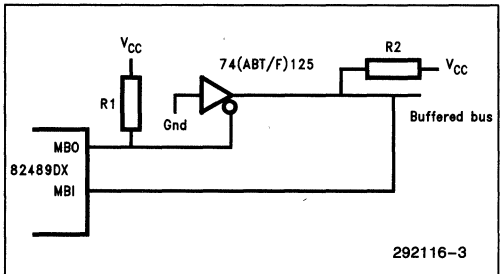
INT3 INT4 INT5 INT6 INT7 INT9 INT10 INT11 INT12 INT14 INT15. In addition to the above 11 interrupts, it fixes **INT8** to be active low edge triggered interrupt. INT8 is the only case where it is active low edge triggered type. So the following logic can be used to add programmability in 82489DX based EISA system. Before connecting these 11 interrupt lines directly (#INT8 which is from Real Time Clock is always active low edge triggered. #INT8 can be passed through an inverter since there is no need for programmability) to the 82489DX they should pass through an array of 11 Ex__OR gates. One input of Ex__OR gate connects to the corresponding INT pin and other input connects to a bit of programmable register. The output of Ex__OR gate is connected to 82489DX. The idea of Ex__OR is to use as a controlled inverter.



INTIN are the interrupt inputs to the 82489DX and INT are the system interrupt. The Ex_OR gating register is programmed after EISA configuration is found from add in boards as how these interrupt lines are going to be used in that particular configuration. If a particular input is edge triggered, then the corresponding bit in the register is written with 0. If a particular input is level triggered, then the corresponding bit in the register is written with 1.

Design Consideration 3

I_{CC} bus drive is an open drain bus with drive capacity of 4 mA only. Since data is transmitted at each I_{CC} clock, the "charging" of I_{CC} bus should be fast enough to ensure proper logic level at each clock edge. The I_{CC} bus needs pull up resistors since it is open drain bus. Since the drive is only 4 mA, the pull up resistor value can not be less than 5V/4mA. This being the limit of the resistor value, the length and the characteristics of the I_{CC} trace forces a capacitance value. Both the resistor and capacitance brings a RC time constant to the I_{CC} bus waveform. So, Electrical consideration has to be given to and practice of controlled impedence should be exercised for layout of the I_{CC} bus. The length of the trace should be kept as minimum as possible. If the length of the I_{CC} bus can't be kept less, than say 6 inch, because of mechanical design of the system, the external line drivers should be added to I_{CC} bus and I_{CC} bus should be simulated with the added driver characteristics.



NOTE:
R1 can be typically 1K.
R2 is designed from the simulation results.

Design Consideration 4

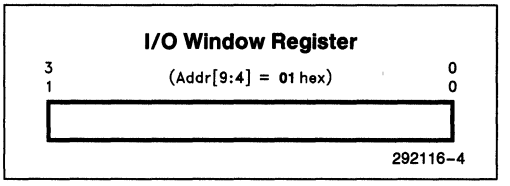
This is related to ADS#, BGT# and CS# timings. For bus cycles not intended for 82489DX, (CS# = 1 where 82489DX is supposed to sample it), any change in CS# line while the ADS# is still active, may erroneously cause a RDY# returned from 82489DX. Anomolous behavior may result if for BGT# ties low cases

a) BGT# goes away just one clock after ADS# or

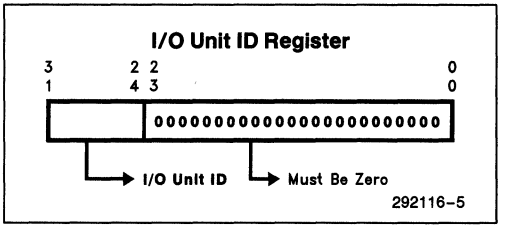
b) ADS# is still active, and CS# changes during this period.

For other cases anomolous behavior results if CS# changes when ADS# is still active. The following considerations are important from timing point of view. Always limit the pulse width of 82489DX ADS# to one CLKIN. Also avoid changing levels on BGT#/CS# line, when ADS# is active for cases being identified as BGT# tied low (BGT# sampled low when ADS# goes active). Also avoid changing levels on CS# line when BGT# is active.

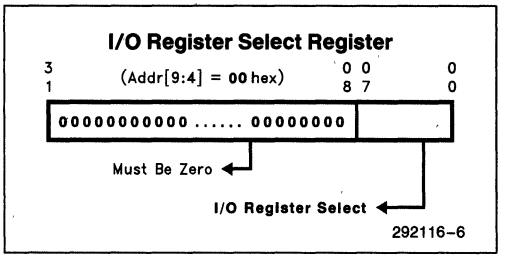
REGISTER PROGRAMMING DETAILS



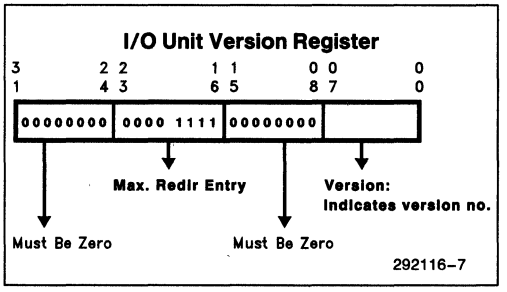
Data access to the register selected by I/O register select Register.



For example, for a Unit ID of 0A hex, the I/O unit ID register should be written with **0A00 0000.**



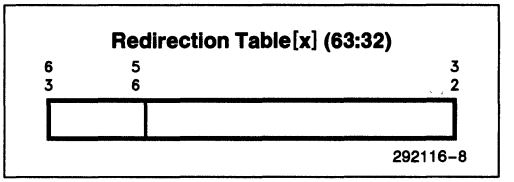
I/O Register Select	Register Selected
00 hex	I/O Unit ID Register
01 hex	I/O Unit Version Register
10 hex	Redirection Table[0] (31:0)
11 hex	Redirection Table[0] (63:32)
12 hex	Redirection Table[1] (31:0)
13 hex	Redirection Table[1] (63:32)
14 hex	Redirection Table[2] (31:0)
15 hex	Redirection Table[2] (63:32)
•	•
•	•
1E hex	Redirection Table[7] (31:0)
1F hex	Redirection Table[7] (63:32)
20 hex	Redirection Table[8] (31:0)
21 hex	Redirection Table[8] (63:32)
•	•
•	•
2E hex	Redirection Table[15] (31:0)
2F hex	Redirection Table[15] (63:32)



I/O Unit Version Register is read only register. It reads as **000F 00XX** where XX is version.

Max.Redir Entry: This is equal to the number of interrupt input pins minus 1 of this I/O unit. Read as 15 in 82489DX.

Version: The version number that identifies this version.



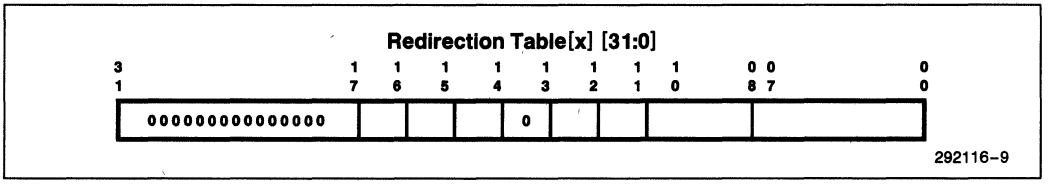
Destination: If the destination mode of this entry is "Physical Mode", then the 8 MSB (bits 56 through 63) contain an 82489DX local unit ID.

If logical mode, then all the 32 bits (bits 63 through 32) of the Destination field potentially defines a set of processors.

NOTE:
The same format holds good for Redirection Tables 0 to 15 (x = 0 to 15) for bits 63 to 32.

If the destination is to a local unit with ID, say, 05 in physical mode, then the redirection table [63:32] should be programmed as hex **0500 0000.**

Redirection Table [63:32] should be programmed first before programming Redirection Table [31:0].



Bits [31:17]: Reserved. Should be written 0.

Bit 16: MASK
0 — Not masked
1 — Masked

Bit [15]: TRIGGER MODE
0 — Edge Triggered
1 — Level Triggered

Bit 14: Remote IRR Status (Read only)
0 — Remote IRR is clear
1 — Remote IRR is set

Bit [13]: Reserved. Should be written 0.

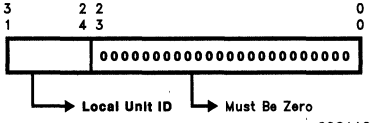
Bit 12: Delivery Status (Read only)
0 — Idle
1 — Send Pending

Bit [11]: Destination Mode
0 — Physical
1 — Logical

Bits [10:8] Delivery Mode
000: Fixed
001: Lowest Priority
100: NMI
101: Reset
111: ExtINTA

Bits [7:0] Vector
Vector for this interrupt

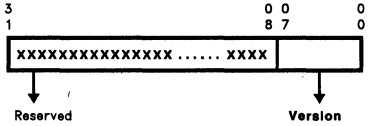
Local Unit ID Register: (Addr (9:4) = hex **02**)



292116-10

For example, for a local Unit ID of 0A hex, the local unit ID register should be written with **0A00 0000.**

Local Unit Version Register
(Addr (9:4) = hex **03**)

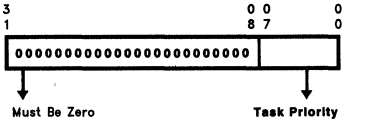


292116-11

Local Unit Version Register is **read only register.** It reads as **0000 00YY** where YY is version number.

Bits [7:0] Version: The version number that identifies this version.

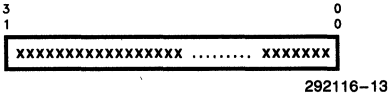
Task Priority Register (Addr (9:4) = hex **08**)



292116-12

Bits [0:7] Task Priority: Should be written with task priority.

End Of Interrupt (EOI) Register
(Addr (9:4) = hex **0B**)

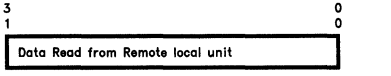


292116-13

Bits [31:0]: Data written to EOI is don't care.

Before returning from the interrupt handler, software must issue an End-Of-Interrupt (EOI) command to the 82489DX local unit. For NMI and ExtINTA and Spurious interrupts EOI SHOULD **NOT** be issued.

Remote Read Register: (Addr (9:4) = hex **0C**)

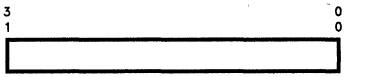


292116-14

The data read from remote local unit is latched in Remote Read Register. The software should qualify this data with "Remote Read Status bit" in the ICR register.

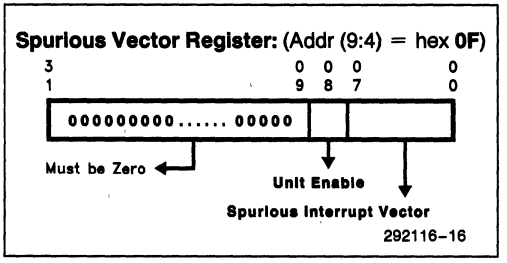
Interrupt Status Register [ISR]:

Register	Address[9:4]
ISR[31:0]	hex 10
ISR[63:32]	hex 11
ISR[95:64]	hex 12
ISR[127:96]	hex 13
ISR[159:128]	hex 14
ISR[191:160]	hex 15
ISR[223:192]	hex 16
ISR[255:224]	hex 17



292116-15

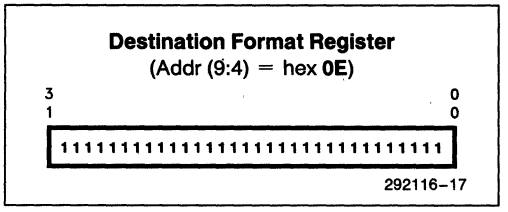
Interrupt Status Register is read only. It marks the interrupts that have been delivered to the processor and waiting for EOI.



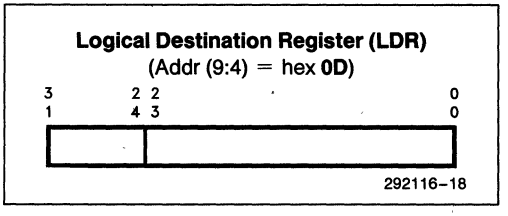
Bits [31:09]: Reserved bits. Must be zero.

Bit 8: Unit Enable: When this bit is 0, the local unit is disabled with regard to transmit and responding messages on I_{CC} bus. It only responds to messages with delivery mode set to "Reset". Reading a 0 at this bit indicates that the unit is disabled. When a 1 is written to the bit, the local unit is enabled for both transmitting and receiving messages. **Once enabled, it should not be disabled by software. Only further resets can take the unit into disabled condition.**

Bits [7:0] Spurious Interrupt Vector: For future compatibility, the bits [3:0] should be written with 1111. A spurious interrupt service routine should be existing in the address corresponding to the spurious interrupt vector.



The destination format register enables logical addressing by specifying the bit map in logical destination register. For future compatibility, all the 32 bits of Destination Format Register should be 1.



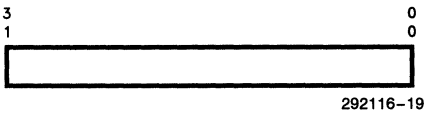
Each local unit can be addressed either physically using physical ID or logically using logical destination register. In physical addressing, either only one local unit can be addressed at a time or broadcast to all local units can be done. In logical addressing, a group of local units can be addressed through bit mapping in destination addressing and the logical destination register.

For future compatibility, bits [0:23] of the logical destination register and bits [0:23] of the destination address in the message should be zero. Bits 24 through 31 of destination information in the interrupt message received are interpreted as decoded field. This field is compared against the logical destination register of the local unit. If there is a bit match (i.e., at least one of the corresponding pair of bits of the destination field and LDR match) that unit is selected for interrupt delivery. Each bit position in the destination field corresponds to an individual Local unit. This scheme allows the specification of arbitrary groups of local units by setting the member's bits to 1, but allows a maximum of 8 local units in a system since only bits 24 through 31 (of the Logical Destination Register and logical destination address in interrupt message) are used. Broadcast to all is achieved by setting all 8 bits of destination to ones. This selects all local units in the system.

In a very large multiprocessor system where future compatibility is not a main problem, all the 32 bits of the Logical Destination Register and all the 32 bits of the destination address in the interrupt message can be used as a bit map to address the processors. When message addresses the destination using logical addressing scheme, the local unit compares the logical address in the interrupt message with its own logical Destination Register. Thus it is possible to support 32 processors in logical addressing mode.

Trigger Mode Register [TMR]:

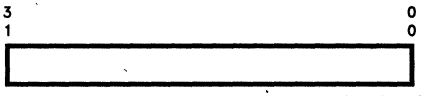
Register	Address[9:4]
TMR[31:0]	hex 18
TMR[63:32]	hex 19
TMR[95:64]	hex 1A
TMR[127:96]	hex 1B
TMR[159:128]	hex 1C
TMR[191:160]	hex 1D
TMR[223:192]	hex 1E
TMR[255:224]	hex 1F



If a bit corresponding to an interrpt vector number is 0, then it is assumed as edge triggered interrupt. For edge triggered interrupt, the corresponding IRR bit is automatically cleared when interrupt service starts. If 1 (level triggered) this is not the case. Instead, the source 82489DX (source I/O unit or Source Local Unit) must explicitly request the IRR bit be cleared (upon deassert of the interrupt input pin or upon sending an appropriate interprocessor interrupt). Upon acceptance of interrupt, the TMR bit is cleared for edge triggered interrupts and set for level triggered interrupts. This information was carried in the accepted interrupt message. The source 82489DX I/O unit also tracks the state of the destination unit's IRR bit (Remote IRR bit in the redirection table). When a level triggered interrupt input is deasserted, the source 82489DX I/O unit detects the discrepancy between the input pin state and the Remote IRR, and automatically sends a message telling destination 82489DX to clear IRR for the interrupt.

Interrupt Request Register [IRR]:

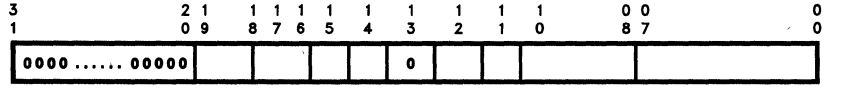
Register	Address [9:4]
IRR[31:0]	hex 20
IRR[63:32]	hex 21
IRR[95:64]	hex 22
IRR[127:96]	hex 23
IRR[159:128]	hex 24
IRR[191:160]	hex 25
IRR[223:192]	hex 26
IRR[255:224]	hex 27



292116-20

It contains the active interrupt requests that have been accepted, but not yet dispensed by this 82489DX local unit. A bit in IRR is set when 82489DX local unit accepts the interrupt. When TMR is 0, it is cleared when the interrupt is serviced; when TMR is 1, it is cleared when the 82489DX local unit receives a message to clear it.

Interrupt Command Register [31:0] (Addr [9:4] = **30** hex)



292116-21

Bits [31:20]: Reserved. Should be written 0.

Bits [19:18]: Destination Shorthand. This field indicates whether a shorthand notation is used to specify the destination of the interrupt and if so, which shorthand is used. Destination shorthands do not use the 32-bit Destination field, and can be sent by software with a single 32-bit write to the 82489DX's interrupt command register. Shorthands are defined for the following cases: Software self interrupt, interrupt to all processors in the system including the sender, interrupts to all processors in the system excluding the sender.

00: (dest field) means that no shorthand is used. The destination is specified in the 32-bit Destination field in the second word (bits 32 to 63) of the interrupt control register.

01: (self) means that the current local unit is the single destination of the interrupt. This is useful for software interrupts. The destination field in the interrupt command register is ignored. RESET assert Delivery mode should not be used with self destination. Only FIXED delivery mode should be used with SELF.

10: (all incl. self) means that the interrupt is to be sent to "all" processors in the system including the processor sending the interrupt. The 82489DX will broadcast a message with destination unit ID field set to all ones. RESET assert Delivery mode should not be used with "all incl. self" destination.

11: (all excl. self) means that the interrupt is to be sent to all processors in the system excluding the processor sending the interrupt. The 82489DX will broadcast a message with destination unit ID field set to all ones.

Bits [17:16]: Remote Read Status. This field indicates the status of the data contained in the Remote Read register. This field is read only to software. Whenever software writes to the interrupt command register using Delivery mode "Remote Read" the Remote Read Status becomes "in progress" (waiting for the remote data to arrive). The remote 82489DX local unit is expected to respond in a fixed amount of time. If the remote 82489DX local unit is unable to do so, then the remote read status becomes "invalid". If successful, the Remote Read status resolves to "Valid". Software should poll this field to determine completion and success of the Remote Read command.

00: (invalid): The content of the Remote Read register is invalid. This is the case when after a Remote Read command is issued and the remote 82489DX Local unit was unable to deliver the Register content in time.

01: (in progress): a remote read command has been issued and this 82489DX is waiting for the data to arrive from remote 82489DX local unit

10: (valid): the most recent Remote Read command has completed and the remote read register content is valid.

11: reserved.

Bit [15]: TRIGGER MODE

0 — Edge Triggered

1 — Level Triggered

Software should use this bit in conjunction with Level Assert/Deassert to generate interrupts that behave as edges or levels. **For future compatibility, send ICR messages only in edge triggered mode.**

Bit [14]: LEVEL. Software should use this bit in conjunction with the Trigger mode bit when issuing an inter-processor interrupt to simulate assertion/deassertion of level sensitive interrupts.

To assert: Trigger mode = 1 and Level = 1.

To deassert: Trigger mode = 1 and Level = 0.

For example, a message with Delivery mode of "Reset", a trigger mode of "Level", and Level bit of 0 deasserts reset to the processor of the addressed 82489DX Local unit(s). As a side effect, this will also cause all 82489DX to reset their Arbitration ID to their unit ID. (The Arb ID is used for tie breaking in lowest priority arbitration.) **For future compatibility, only edge triggering should be used in ICR.**

Bit [13]: Reserved. Should be written 0.

Bit [12]: Delivery Status (Read only)

0 — Idle

1 — Send pending

Delivery status is software read-only. Software can read to find out if the current interrupt has been sent, and the Interrupt command register is available to send the next interrupt. If the interrupt command register is overwritten before the Delivery status is "idle", then the destiny of that interrupt is undefined; the interrupt may have been lost.

Bit [11]: Destination Mode

0 — Physical

1 — Logical

In physical mode, a destination 82489DX is identified by its Local Unit ID. Bits 56 through 63 (8 MSB of the destination field) specify the 8-bit 82489DX Local unit ID.

In logical mode, destinations are identified by matching on Logical Destination under the control of the Destination Format Register in each Local 82489DX. The 32-bit Destination field is the logical destination. **For future compatibility, use only bits [31:24] of the logical destination address.** Bits [23:0] should be zero.

Bits [10:8]: Delivery Mode

000: (Fixed) means deliver the signal on the INT pin of all processors listed in the destination. Trigger mode for "fixed" Delivery Mode can be edge or level.

001: (Lowest Priority) means deliver the signal on the INT pin of the processor that is executing at the lowest priority among all the processors listed in the specified destination; Trigger mode for "lowest priority" Delivery mode can be edge or level.

011: (Remote Read) is a request to a remote 82489DX local unit to send the value of one of its registers over the I_{CC} bus. The register is selected by providing its address in the vector field. The register value is latched by the requesting 82489DX and stored in the Remote Register where it can be read by the local processor. A Delivery Mode of "Remote Read" requires an "Edge" Triggered mode.

100: (NMI) means deliver the signal on the NMI pin of all processors listed in the destination. Vector information is ignored. A delivery mode equal to "NMI" requires a "LEVEL" Trigger mode.

101: (Reset) means deliver the signal to all processors listed in the destination by asserting/deasserting the 82489DX local unit's PRST output pin. All addressed 82489DX local units will assume their reset state but preserve their ID. One side effect of a message with Delivery mode equal to "Reset" that results in a deassert of reset is that all Local Units (whether listed in the destination or not) will reset their lowest-priority tie breaker arbitration ID to their Local unit ID. A delivery mode of "Reset" requires a "level" Trigger mode. "Reset" should not be used with "Self" or "all incl.Self" Shorthand mode since it will leave the system in non-recoverable reset state. If "RESET" is used with "all exc.Self" mode, software should make sure that only one CPU executes this instruction in an MP system.

Delivery mode options **010,110,111** are Intel reserved. They should not be used.

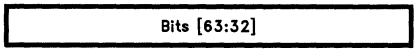
Bits [7:0] Vector. The vector identifies the interrupt being sent. If the Delivery mode is "Remote Read", then the Vector field contains the address of the register to be read in the remote 82489DX's Local unit.

NOTE:
In cases where Destination field in Interrupt Command Register [63:32] is used, Interrupt Command Register [31:0] should be programmed only **AFTER** programming Interrupt Command Register [63:32], since writing to [31:0] will start sending the message.

The following are the control words for interrupt command register [31:0] for different modes. The interrupt vector, for example, is illustrated with **AA hex**. In the remote Read request command **RR in the vector field** specify address of the register to be read. The **XX** in the vector field means the vector is don't care.

CONTROL WORD	PHYSICAL Destination Mode	LOGICAL Destination Mode
Fixed INT, Edge triggered int, *dest. field specified*	0000 00AA hex	0000 08AA hex
Lowest priority INT, Edge trigg. int, *dest. field specified*	0000 01AA hex	0000 09AA hex
Remote Read (only Edge Triggered), *dest. field specified*	0000 03RR hex	NOT SUPPORTED
NMI (Only Level) Level ASSERT, *dest. field specified*	0000 C4XX hex	0000 CCXX hex
NMI (Only Level) Level DEASSERT, *dest. field specified*	0000 84XX hex	0000 8CXX hex
Reset (Only Level) Level ASSERT, *dest. field specified*	0000 C5XX hex	0000 CDXX hex
Reset (Only Level) Level DEASSERT, *dest. field specified*	0000 85XX hex	0000 8DXX hex
Fixed INT, Edge triggered int, *Self*	0004 00AA hex	0004 08AA hex
Fixed INT, Edge trigg. int, *All inclusive Self*	0008 00AA hex	0008 08AA hex
Lowest priority INT, Edge trigg. int, *All inclusive Self*	0008 01AA hex	0008 09AA hex
NMI, Level ASSERT, *All inclusive Self*	0008 C4XX hex	0008 CCXX hex
NMI, Level DEASSERT, *All inclusive Self*	0008 84XX hex	0008 8CXX hex
Reset, Level DEASSERT, *All inclusive Self*	0008 85XX hex	0008 8DXX hex
Fixed INT, Edge trigg. int, *All exclusive self*	000C 00AA hex	000C 08AA hex
Lowest priority INT, Edge trigg. int, *All exclusive self*	000C 01AA hex	000C 09AA hex
NMI, Level ASSERT, *All exclusive Self*	000C C4XX hex	000C CCXX hex
NMI, Level DEASSERT, *All exclusive Self*	000C 84XX hex	000C 8CXX hex
Reset, Level ASSERT, *All exclusive Self*	000C C5XX hex	000C CDXX hex
Reset, Level DEASSERT, *All exclusive Self*	000C 85XX hex	000C 8DXX hex

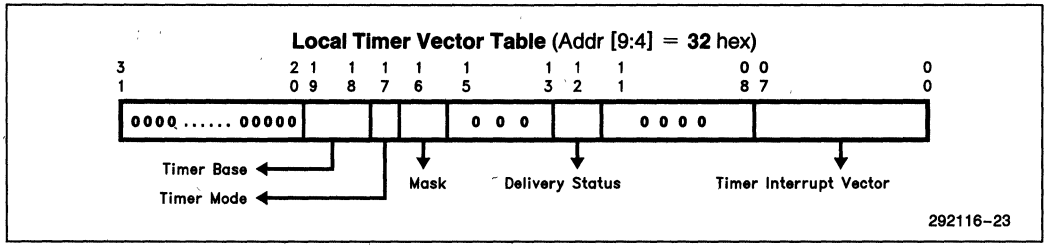
Interrupt Command Register [63:32]
(Addr [9:4] = **31** hex)



292116-22

Bits [63:32] Destination

This field is only used when the Destination Shorthand field is set to "Destination Field". If Destination field is physical mode, **then the 8 MSB contain an Destination Unit ID.** If logical mode, the full 32-bit Destination field contains the logical address. This register should be programmed for proper destination before programming Interrupt Command Register [31:0]. If the destination to a local unit with ID, say, 05 in physical mode, then the Interrupt Command Register [63:32] should be programmed as hex **0500 0000.**



Bits [31:20] Reserved. Should be written Zero.

Bits [19:18] Timer Base: This field selects the time base input to be used by timer.

00: (Base 0): Uses "CLKIN" as input.

01: (Base 1): Uses "TMBASE".

10: (Base 2): Uses the output of the divider (Base 2).

Bit 17: Timer Mode: This field indicates the operation mode of timer.

0 — ONE- SHOT;

1 — PERIODIC

In *ONE-SHOT*, the current count register remains at Zero after the timer reaches zero and software needs to reassign the timer's initial count register to rearm the timer.

In *PERIODIC* mode, when the timer reaches zero, the Current Count Register is automatically reloaded with the value in the initial Count Register, and the timer counts down again.

Bit 16: Mask: This bit serves to mask timer interrupt generation.

0 — Not masked;

1 — Masked.

Bits [15:13] Reserved. Should be written Zero.

Bit [12] Delivery Status: Delivery status indicates the current status of the delivery status of this interrupt.

0 — IDLE means that there is currently no activity for this interrupt.

1 — SEND PENDING indicates that the interrupt has been injected, but its delivery is temporarily held up by other recently injected interrupts that are in the process of being delivered; Delivery status is software read only.

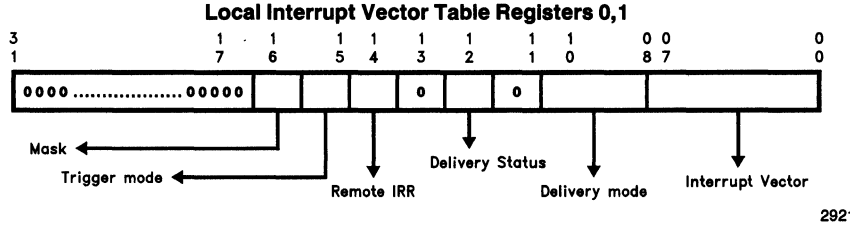
Bits [11:8] Reserved. Should be written Zero.

Bits [7:0] Timer Interrupt vector: This is the 8-bit interrupt vector to be used when timer generates an interrupt.

NOTE:
TIMER interrupts are always treated as EDGE triggered interrupts.

The following is the control word for various modes to be used in Local Timer Vector Table. For illustration purpose, the interrupt vector for Timer is shown as AA hex.

Control Word	CLKIN Input (Base 0)	TMBASE Input (Base 1)	Divider Input (Base 2)
PERIODIC timer, MASK cleared	0002 00AA hex	0006 00AA hex	000A 00AA hex
PERIODIC timer, MASK set	0003 00AA hex	0007 00AA hex	000B 00AA hex
ONE SHOT timer, MASK cleared	0000 00AA hex	0004 00AA hex	0008 00AA hex
ONE SHOT timer, MASK set	0001 00AA hex	0005 00AA hex	0009 00AA hex



Register	Address [9:4]
Local Int0 Vector table register	**35** hex
Local Int1 Vector table register	**36** hex

The same format applies to both Local Int0 and Local Int1 registers.

Bits [31:17]: Reserved: Must be Zero.

Bit 16: MASK:

0 — enables interrupt by clearing mask

1 — masks the interrupt.

Bit 15: Trigger mode:

0 — Edge Triggered

1 — Level Triggered

Bit 14: Remote IRR: This bit is used for level triggered local interrupts. Its meaning is undefined for edge triggered interrupts. Remote IRR mirrors the interrupt's IRR bit of this local unit. Remote IRR is software read only.

Bit 13: Reserved. Must be Zero.

Bit 12: Delivery Status: Software read only. Indicates the current status of the delivery of this interrupt.

0 — **IDLE** means that there is currently no activity for this interrupt.

1 — **Send Pending** indicates that the interrupt has been injected, but its delivery is temporarily held up by the recently injected interrupts that are in the process of being delivered.

Bit 11: Reserved. Must be Zero.

Bits [10:8]: Delivery mode

000 — Fixed INT

100 — NMI

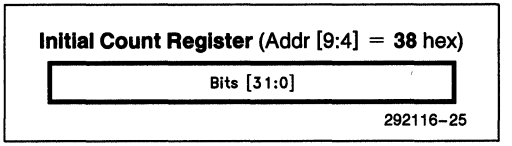
111 — ExtINTA

All other options of Bits [10:8] are reserved. Should not be used.

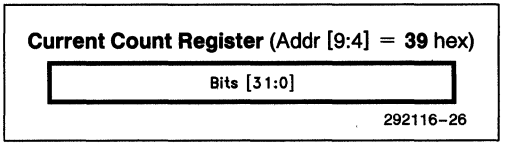
Bits [7:0]: Vector: This is the interrupt vector to use when generating interrupt for this entry.

The following are the control words for local interrupt [0 as well as 1] vector tables for different modes. The interrupt vector, for example, is illustrated with **AA hex**. The XX in the vector field means the vector is don't care.

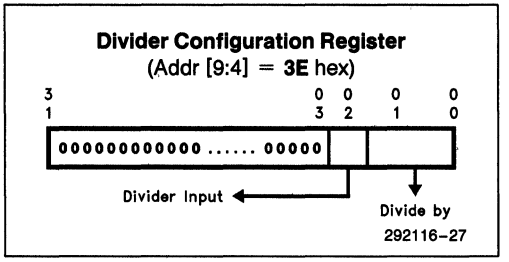
Interrupt Option	Control Word
Fixed INT, Edge triggered	0000 00AA hex
Fixed INT, Level trigg. int	0000 80AA hex
NMI (Only Level)	0000 84XX hex
ExtINTA (Only Edge)	0000 07XX hex



Bits [31:0] Initial Count: Software writes to this register to set the initial count for timer. This register can be written at any time. When written, the value is copied to the current count Register and countdown starts or continues from there. The initial count register is read-write by software.



Bits [31:0] Current Count: This is the current count of timer. It is read only by software and can be read any time.



Configuration	Control Word
Divide CLKIN by 2	0000 0000 hex
Divide CLKIN by 4	0000 0001 hex
Divide CLKIN by 8	0000 0002 hex
Divide CLKIN by 16	0000 0003 hex
Divide TMBASE by 2	0000 0004 hex
Divide TMBASE by 4	0000 0005 hex
Divide TMBASE by 8	0000 0006 hex
Divide TMBASE by 16	0000 0007 hex

Bits [31:3] Reserved. Must be Zero.

Bit [2]: Divider Input: Selects whether divider's input connects to the 82489DX local unit's CLKIN pin or TMBASE.

0 — means the divider takes its input signal from CLKIN.

1 — means use TMBASE

Bits [1:0]: Divide by: Selects by how much the divider divides.

00 — divide by 2

01 — divide by 4

10 — divide by 8

11 — divide by 16

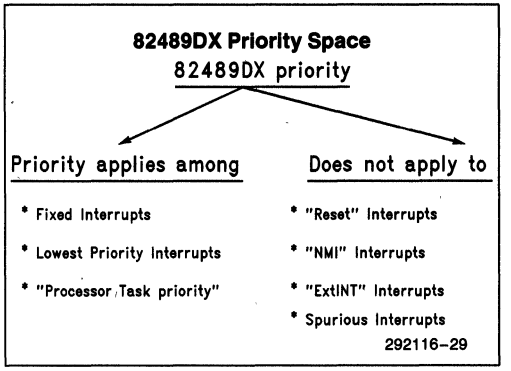
Programming Guidelines

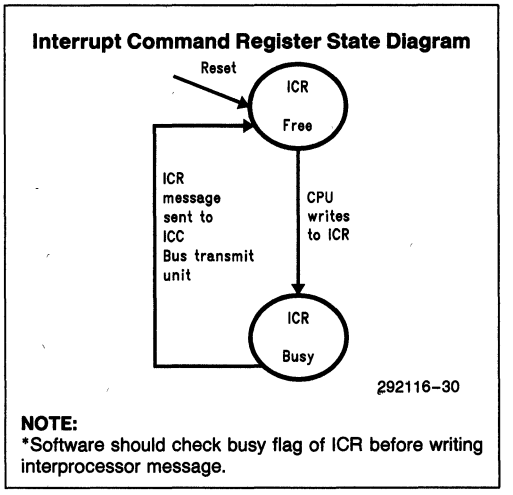
A) Modes of Interrupt in 82489DX:

Trigger Mode	Delivery Mode				
	Fixed Destination	Lowest Priority Delivery	NMI	Reset	ExtINT
Edge	✓	✓			✓
Level	✓	✓	✓	✓	

NOTE:

- RESET delivery mode should not be used for Local Interrupts.
- EOI should not be issued for NMI and ExtINT delivery mode.





CONCLUSION

82489DX has simple and powerful programming model. It has programmable priority and it supports task priority in the light of interrupt priority. It reduces the SPL() overhead which is very useful in uniprocessor system. The system performance is improved by using interrupt priority model to prioritize interrupts and by using task priority register for SPL() calls. It provides an easy migration path from 8259 by providing ExtINTA mode for DOS compatibility.